

Lecture 2

C# Data structure oriented warming:

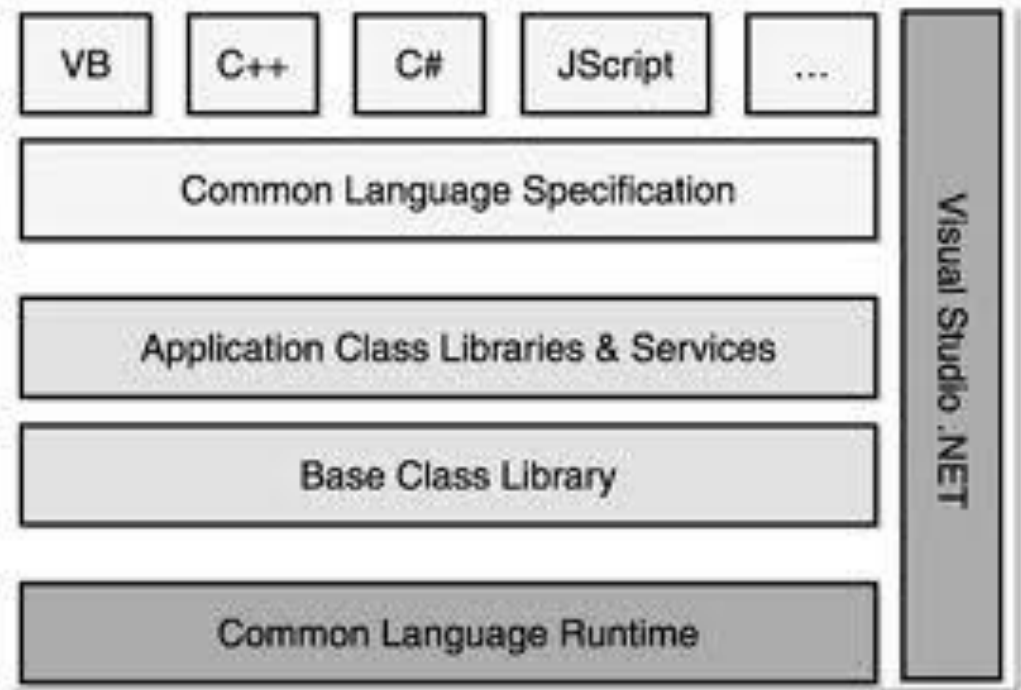
Arrays

Generic programming

Lists

The .NET framework

- The framework is a combination of methodologies and tools to support software development
- It's lowest level component is the CLR which interacts directly with the operating system and the native world
- Application developer writes his application using one of the supported languages
- All supported language obey the common language specifications
- Application class library and base class library are ready to use customizable parts
- Some common data structures classes are made available in .NET



C# program structure and classes

```
using System;
namespace Introduction
{
}

```

```
61 public class Student1Test
62 {
63     static void Main ()
64     {
65         Console.WriteLine("Hello to the student class program");
66         // using the parameterized constructor
67         Console.WriteLine("Student created up to now=" + Student.StudentCreatedCount.ToString());
68         Student st1 = new Student("Ahmed", "Hamed", "Hemeda");
69         st1.Marks = 90;
70         Console.WriteLine(st1.GetFullName() + " grade is :" + st1.GetGrade());
71         Console.WriteLine("Student created up to now=" + Student.StudentCreatedCount.ToString());
72         // the parameter less constructor
73         Student st2 = new Student();
74         st2.Marks = double.Parse(Console.ReadLine());
75         Console.WriteLine(st2.GetFullName() + " grade is :" + st2.GetGrade());
76         Console.WriteLine("Student created up to now=" + Student.StudentCreatedCount.ToString());
77         Console.WriteLine("Press any key to continue:");
78         Console.Read();
79     }
80 }

```

```
6 class Student
7 {
8     public static int StudentCreatedCount = 0;
9     // instance data
10    private string firstName;
11    private string middleName;
12    private string lastName;
13
14    private double marks;
15    //parameterized constructor
16    public Student(string firstName, string middleName, string lastName)
17    {
18        this.firstName = firstName;
19        this.middleName = middleName;
20        this.lastName = lastName;
21        StudentCreatedCount++;
22    }
23    // parameter-less constructor
24    public Student()
25    {
26        StudentCreatedCount++;
27    }
28    public double Marks
29    {
30        get { return this.marks; }
31        set { this.marks = value; }
32    }
33    public string GetGrade()
34    {
35        if (this.marks < 50)
36        {
37            return "Fail";
38        }
39        else if (this.marks < 65)
40        {
41            return "Pass";
42        }
43        else if (this.marks < 75)
44        {
45            return "Good";
46        }
47        else if (this.marks < 85)
48        {
49            return "Very Good";
50        }
51        else
52        {
53            return "Excellent";
54        }
55    }
56    public string GetFullName()
57    {
58        return this.firstName+" "+this.middleName+" "+this.lastName;
59    }
60 }

```

One dimensional arrays in C#

```
using System;
namespace Introduction
{
    class Arrays1
    {
        static void Main()
        {
            // creating the array
            int[] numbers = new int[5] { 0, 1, 2, 3, 4 };
            // printing the array to the screen
            Console.WriteLine(" the array elements are:");
            for (int i = 0; i < numbers.Length; i++)
                Console.WriteLine(numbers[i]);
            // changing the values in the array
            numbers[0] = 5;
            numbers.SetValue(6, 1);
            numbers.SetValue(7, 2);
            numbers[3] = 8;
            numbers[4] = 9;
            // representing the array elements
            Console.WriteLine(" The array elements after updating are:");
            for (int i = 0; i < numbers.GetLength(0); i++)
                Console.WriteLine(numbers[i]);
            // representing the array elements using get upperBound
            Console.WriteLine(" The array elements after updating are (using GetUpperBound method):");
            for (int i = 0; i <= numbers.GetUpperBound(0); i++)
                Console.WriteLine(numbers[i]);
            // printing the array type on the screen
            Console.WriteLine("the Array type is" + numbers.GetType().ToString());
            Console.WriteLine("Press any key to continue:");
            Console.Read();
        }
    }
}
```

The array class methods and properties

```
using System;
namespace Introduction
{
    class Arrays1
    {
        static void Main()
        {
            // creating the array
            int[] numbers = new int[5] { 0, 1, 2, 3, 4 };
            // printing the array to the screen
            Console.WriteLine(" the array elements are:");
            for (int i = 0; i < numbers.Length; i++)
                Console.WriteLine(numbers[i]);
            // changing the values in the array
            numbers[0] = 5;
            numbers.SetValue(6, 1);
            numbers.SetValue(7, 2);
            numbers[3] = 8;
            numbers[4] = 9;
            // representing the array elements
            Console.WriteLine(" The array elements after updating are:");
            for (int i = 0; i < numbers.GetLength(0); i++)
                Console.WriteLine(numbers[i]);
            // representing the array elements using get upperBound
            Console.WriteLine(" The array elements after updating are (using GetUpperBound method):");
            for (int i = 0; i <= numbers.GetUpperBound(0); i++)
                Console.WriteLine(numbers[i]);
            // printing the array type on the screen
            Console.WriteLine("the Array type is" + numbers.GetType().ToString());
            Console.WriteLine("Press any key to continue:");
            Console.Read();
        }
    }
}
```

Warning: this is a draft copy. It has not been passed any revision

Multi-Dimensions array

Syntax

```
int[,] grades = new int[4,5];  
double[,] Sales;  
double[,] sales;  
sales = new double[4,5];  
grade = Grades[2,2];  
Grades(2,2) = 99  
grade = Grades.GetValue[0,2]
```

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace Lecture2  
{  
    class Example3  
    {  
        static void Main()  
        {  
            int[,] grades = new int[,] {{77, 82, 74, 89, 100},  
                {52, 93, 96, 85, 86},  
                {65, 83, 72, 95, 89},  
                {55, 91, 98, 79, 88}};  
            int gradeCount = grades.GetLength(1);  
            double average = 0.0;  
            int total;  
            int studentCount = grades.GetLength(0);  
            for (int row = 0; row < studentCount; row++)  
            {  
                total = 0;  
                for (int col = 0; col < gradeCount; col++)  
                {  
                    total += grades[row, col];  
                    average = total / gradeCount;  
                    Console.WriteLine(" student {0} Average grade:{1} ",row,average);  
                }  
                Console.WriteLine("Press any key to continue:");  
                Console.Read();  
            }  
        }  
    }  
}
```

Parameter array

```
using System;

namespace Lecture2
{
    class Example4
    {
        static double Sum(params double[] values)
        {
            double sum = 0;
            for(int i=0;i<values.Length;i++)
                sum+=values[i];
            return sum;
        }
        static void Main()
        {
            Console.WriteLine(" The sum of values from 0 to 5 is:" + Sum(0, 1, 2, 3, 4, 5));
            Console.WriteLine("Press any key to continue:");
            Console.Read();
        }
    }
}
```

Jagged array

Multi-dimensions array(2D): Each row must have the same number of columns
Example: `int sales[,] = new int[12,30];` for representing day sales during the year
This is also applicable for more than 2D

But some months have 31 days and others have 28 days
So, there will be some location unused in the multi-dimensions array

Jagged array(2D): Each row can have different number of columns(array of arrays)
Example: `int[][] jagged = new int[12][];` 12 array, each can be of any size int array
This is also applicable for more than 2D

Jagged array: 12 arrays, each can have different size than the others.
Hence, no wasted locations

Jagged array example

```
using System;

namespace Lecture2
{
    class Example5
    {
        static void Main()
        {
            int[] Jan = new int[31];
            int[] Feb = new int[29];
            int[][] sales = new int[][] { Jan, Feb };
            // data entry
            for (int i = 0; i < sales.GetLength(0); i++)
            {
                for (int j = 0; j < sales[i].Length; j++)
                {
                    Console.WriteLine("\n Enter sales for month {0} Day {1}:", i, j);
                    sales[i][j] = int.Parse(Console.ReadLine());
                }
            }
            // average sales calculation
            int monthTotalSales;
            for (int i = 0; i < sales.GetLength(0); i++)
            {
                monthTotalSales=0;
                for (int j = 0; j < sales[i].Length; j++)
                {
                    monthTotalSales += sales[i][j];
                }
                Console.WriteLine(" month {0} average sales is : {1}", i, monthTotalSales / sales[i].Length);
            }
            Console.WriteLine("Press any key to continue:");
            Console.Read();
        }
    }
}
```

Passing arrays to functions

```
using System;
namespace Lecture2
{
    class Example6
    {
        static void DisplayIntArray(int[] array)
        {
            Console.WriteLine("The elements of the array are as follows:");
            for (int i = 0; i < array.Length; i++)
            {
                Console.WriteLine(" Element {0} value is {1}", i, array[i]);
            }
        }
        static void Main()
        {
            int[] array1 = new int[] { 1, 2, 3, 4, 5 };
            DisplayIntArray(array1);
            int[] array2 = new int[] { 6, 7, 8, 9, 10 };
            DisplayIntArray(array2);
            Console.WriteLine("Press any key to continue:");
            Console.Read();
        }
    }
}
```

Generic programming in c#: Functions

A generic function is defined using data type place holders like T in the swap function

When calling a generic function, we specify the specific data types for this call. At this time the compiler generate a type specific function at the compile time. It uses this generated function for that call and any matching call after that.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Lecture2
{
    class Example7
    {
        static void Main()
        {
            int num1 = 100, num2 = 200;
            Console.WriteLine("num1: " + num1);
            Console.WriteLine("num2: " + num2);
            Swap<int>(ref num1, ref num2);
            Console.WriteLine("num1: " + num1);
            Console.WriteLine("num2: " + num2);
            string str1 = "Sam";
            string str2 = "Tom";
            Console.WriteLine("String 1: " + str1);
            Console.WriteLine("String 2: " + str2);
            Swap<string>(ref str1, ref str2);
            Console.WriteLine("String 1: " + str1);
            Console.WriteLine("String 2: " + str2);
            Console.WriteLine("Press any key to continue:");
            Console.Read();
        }

        static void Swap<T>(ref T val1, ref T val2)
        {
            T temp;
            temp = val1;
            val1 = val2;
            val2 = temp;
        }
    }
}
```

Generic programming in c#: Classes

A List(of T) is a dynamic array of T> You can add element or elements to the list at any time. Unlike array which is of fixed size once created, the list automatically enlarges to make a room for the added elements. You can get how many elements in the List using the Count property. You can even sort the list using the Sort method

```
using System;
using System.Collections.Generic;
namespace Lecture2
{
    class Example8
    {
        static void Main()
        {
            List<int> intList = new List<int>();
            intList.Add(5);
            intList.AddRange(new int[]{5,6,7});
            Console.WriteLine(" the integerList contains {0} items", intList.Count);
            intList.Clear();
            Console.WriteLine(" the integerList contains {0} items", intList.Count);
            List<string> names = new List<string>();
            names.Add("Ahmed");
            names.Add("Ali");
            Console.WriteLine("Printing the names list:");
            foreach (string name in names)
            {
                Console.WriteLine(name);
            }
            Console.WriteLine("Press any key to continue:");
            Console.Read();
        }
    }
}
```

Report Discussion

Last lecture report: Performance
comparison: iterative and recursive binary
search

New report: .NET List(of T) methods and
properties